Laplacian Texture Synthesis and Mixing on Surfaces

Qing Wu   Lin Shi   Stephen Bond   Yizhou Yu

Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

In neighborhood-based texture synthesis, adjacent local regions need to satisfy color continuity constraints in order to avoid visible seams. Such continuity constraints seriously restrict the variability of synthesized textures, making it impossible to generate new textures by mixing multiple input textures with very different base colors. In this paper, we propose to relax such restrictions and decompose synthesis into two relatively disjoint stages. In the first stage, an intermediate synthesized texture is generated by only considering the high frequency details during region search and matching. Such a scheme broadens the search space during texture synthesis, but may produce obvious seams due to large discontinuities in low frequency components. In the second stage, instead of performing local feathering along these discontinuities, we perform Laplacian texture reconstruction, which retains the high frequency details but computes new consistent low frequency components to eliminate the seams. It does not only affect texels close to the discontinuities, but also modifies the rest of the texels. Therefore, it can be viewed as a global feature-preserving smoothing step, and is more effective than local feathering. Experiments indicate that our two-stage synthesis can produce desirable results for regular texture synthesis as well as texture mixing from multiple sources.

1 Introduction

Texture synthesis has been widely recognized as an important research topic in computer graphics. Early texture synthesis algorithms were based on global statistical models [9, 18, 30]. Instead of enforcing global statistics, preserving the local arrangement of pixels has proven to be more effective in terms of visual quality. This intuition led to the recent neighborhood-based search-and-copy algorithms [7, 24, 1, 13, 10] and their optimized versions [6, 12]. Given a small texture example, these algorithms can produce larger textures that have similar texture elements and structures as the given example. Every output texture from such neighborhood-based texture synthesis is essentially a spatial rearrangement of the original local regions in the given example. When there is only a single small texture example, the number of possible rearrangements is actually limited because adjacent local regions need to satisfy continuity restrictions.

We propose to relax such neighborhood-based texture synthesis along two directions to improve the variability of the synthesized results without compromising their visual quality. First, relax the continuity restrictions. Previously, adjacent local regions in the output texture are typically required to have pixelwise color similarity in their overlapping portion. When a new local region needs to be chosen from the texture example, such a stringent condition results in a very small number of candidates and quite often zero candidates. We suggest to relax such region matching by focusing on the high frequency components only and overlooking the average color and intensity. Indeed, it is the high frequency components that play the most important role in characterizing a texture.

Second, allow multiple input texture examples. Sampling local regions from a single small texture example can only produce very limited variability. Ideally, example-based synthesis should generate results by sampling a large database. However, different textures may be acquired by different imaging devices and/or settings, under different illumination conditions, etc. Even the same real-world texture, such as grass, can appear very different in different texture images. Effectively sampling, matching and mixing local regions from multiple texture examples simultaneously is nontrivial.

In this paper, we focus on surface texture synthesis and propose a novel two-stage synthesis approach to accommodate these two relaxations. In the first stage, an intermediate synthesized texture is generated by only considering the high frequency details during region search and matching. It is achieved by using both features and rectified versions of the input textures. Each pixel value in the rectified textures is defined by its original value normalized by the accumulated intensity within a neighborhood. Such a scheme broadens the search space during texture synthesis. It facili-

tates sampling local regions from different texture examples as well as placing regions with very different average colors and intensities next to each other in the output texture. However, this intermediate texture has obvious seams due to large discontinuities in low frequency components.

In the second stage, to seamlessly mix local regions together and create smooth transitions among them, we perform Laplacian texture reconstruction which is in the same spirit of Poisson image editing [16] and related surface editing [20, 27]. It retains the high frequency details, but computes new consistent low frequency components for all local regions so that the seams among them disappear. Therefore, it can be viewed as a global feature-preserving smoothing step, and is more effective than local feathering. The Laplacian operator extracts local differential quantities which represent high frequency texture details. Given the Laplacian of the intermediate texture, we set up a sparse linear system with the new texture colors as the unknowns. The solution of this system not only retains the original texture details, but also provides a consistent coloring to all the texels in the synthesized texture without discontinuities along the boundaries of adjacent regions.

1.1 Related Work

This paper is partially inspired by the recent success of texture synthesis [9, 4, 18, 30, 7, 24, 2, 1, 13, 6, 10, 5, 29, 12, 14, 26]. In 2D texture synthesis, many algorithms model textures as Markov Random Fields and generate textures by probability sampling [30, 7, 24]. Some algorithms model textures as a set of features, and generate new images by matching feature statistics, such as histograms and co-occurrences, potentially across multiple resolutions [9, 4, 18, 2]. Recently, patch-based texture synthesis [13, 6, 12, 26] achieved better results than the previous methods in terms of both quality and efficiency. Feature continuity on the boundary of two adjacent patches is an important issue and [6, 12, 26] have attempted to alleviate this problem using dynamic programming, graph cuts, and feature maps, respectively.

There have also been quite a few works [22, 25, 19, 29, 15, 3, 28] on generalizing 2D texture synthesis onto meshes with arbitrary topology. The method in [24] was generalized to meshes in [22, 25], which perform hierarchical vertex-based synthesis. A binary texton mask was introduced in [29] as guidance data to improve synthesis results and reduce the number of broken features. Patch-based synthesis has also been generalized to meshes [19, 15]. A hierarchical patch-based approach was presented in [19] while Magda and Kriegman [15] introduced an efficient synthesis method on triangle meshes. A very fast synthesis technique accelerated by precomputed Jump Maps was introduced in [28]. Neighborhood or patch matching in these methods is

directly based on color differences instead of differences in high frequency components. In [3], texture synthesis was further generalized to geometric detail synthesis over mesh surfaces.

Meanwhile, researchers have synthesized textures from multiple input examples by mixing together different elements from them [9, 18, 2, 23, 29]. In [9, 18], texture mixtures are synthesized in the domain of steerable pyramids. In [2], statistical learning trees are used to mix textures. In [23], new texels in a synthesized mixture are grown by looking at the distances to all the input examples simultaneously. On the other hand, the technique in [29] focuses on creating a progressive transition between different texture elements. However, these texture mixing techniques cannot globally adjust the colors of the mixed textures to make them more consistent with each other.

2 Overview

The input to our algorithm is a triangle mesh as well as one or more texture examples. The output is the same mesh covered with a texture synthesized from the given examples. In the case of multiple input textures, the synthesized texture is a spatial mixture of the texture elements from the texture examples.

Our Laplacian texture synthesis algorithm has three basic steps.

1. Apply a revised version of the method in [15] to generate an initial texture patch assignment on the mesh. The method in [15] does not directly synthesize textures on a mesh. Instead, it assigns a triangular texture patch in the input textures to each triangle in the mesh. The assigned texture patches of two adjacent triangles have a certain degree of continuity along their shared edge. Our revised version tries to emphasize high frequency details, but overlook the differences in the average colors of local regions. At the end of this step, each triangle in the mesh is associated with three pairs of texture coordinates which record the locations of the corners of its corresponding triangular texture patch in the input texture examples.

2. Each triangle in the mesh is tesselated with a high-resolution grid and the assigned texture patch of the triangle is resampled onto this grid. A graphcut algorithm is further executed to refine the boundary between two adjacent texture patches so that such a boundary is not necessarily coincidental with the shared edge between two adjacent triangles any more. As a result, the transitions of details among texture patches are improved though their average colors may still be quite different.
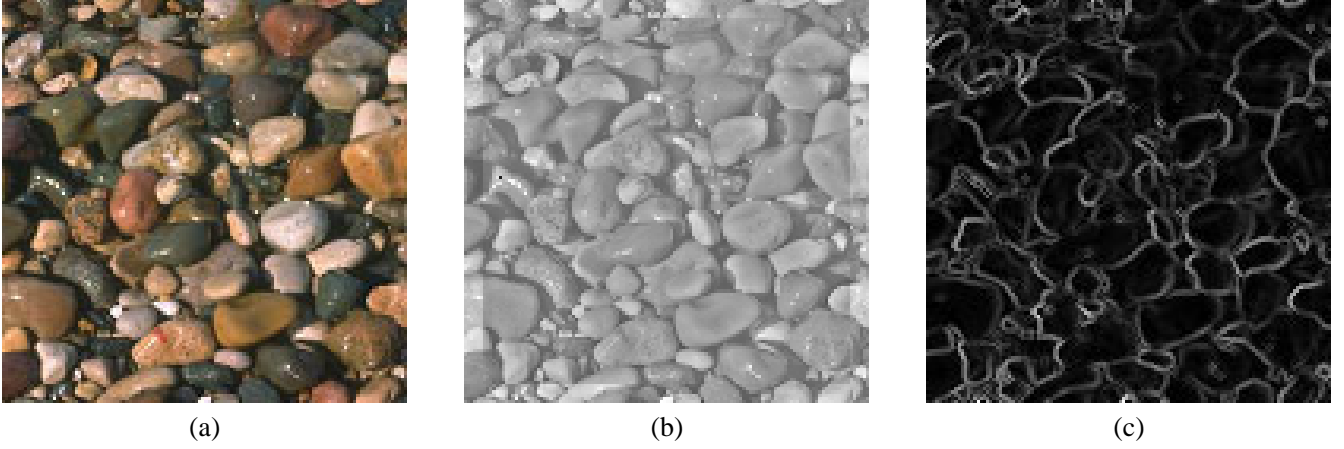
Figure 1. (a) The original PEBBLES **texture. (b) The rectified greyscale image of (a). (c) A feature image of (a) obtained by filtering.**

3. Laplacian texture reconstruction is performed simultaneously on all the resampled texture patches from the previous step to eliminate the color discontinuities between adjacent patches. The Laplacian at each grid point is obtained from the original colors in the input texture examples.

3   Initial Texture Assignment

Our initial texture assignment is based on the method in [15], where a texton is defined to be a distinct local texture neighborhood. By clustering all neighborhoods with a fixed size from the given texture example, a small collection of textons can be extracted. They are the representatives of the clusters. During synthesis, triangular texture patches are grown on the mesh one by one to cover all the triangles. Note that each triangle in the mesh shares an edge with at most three adjacent triangles. During each step of synthesis, this method focuses on one triangle and counts the number of its adjacent triangles that have been covered with texture. If none of them has been covered, the current triangle is a seed and should be covered with a random patch. If one of them has been covered, we need to search for a texture patch in the given texture example that agrees well with the texture on this adjacent patch, which means that the two texton sequences on the shared edge should be similar. This strategy can be easily generalized to cases where two or three adjacent triangles are already covered with textures.

To emphasize high frequency details but overlook differences in average intensity and color, in our revised version of this method, we utilize a rectified version of each input texture. The rectified version of a texture is a greyscale image with a normalized intensity value at each pixel. We set the initial greyscale value at a pixel to be its luminance value which is a weighted average of the original three color channels. Suppose we define an $N \times N$ neighborhood for each pixel and the luminance at a neighbor (i, j) is I_{ij}. The luminance value at each pixel is further normalized by the accumulated luminance in its neighborhood, which is $(\sum_{ij} I_{ij}^2)^{1/2}$. In this rectified texture, we essentially have removed the low frequency components, but retained the important high frequency details. We only use greyscale values because they are weighted averages of three color channels and contain high frequency details from all of them. In practice, using greyscale values indeed can produce better matching results than using three independent color channels. An example of a rectified texture is shown in Fig. 1(b). In practice, we set the size of the neighborhoods to be 11×11.

In addition to the rectified textures, we also obtain a feature image for each of the input texture examples. We first apply bilateral filtering [21] to remove noise while preserving features. In the bilateral filter, the scale of the closeness function σ_d is set to 2.0, and the scale of the similarity function σ_r is set to 10 out of 256 greyscale levels. We then use finite differences along the two image axes as a simple gradient estimator to obtain an edge response at every pixel. The pixelwise gradient estimation is used to form the feature images. An example of a feature image is shown in Fig. 1(c).

In our revised version of the method in [15], we use these rectified textures along with the feature images as the input to texton clustering. Thus, the neighborhood corresponding to each texton has a normalized greyscale pattern and a feature pattern. Both patterns emphasize high frequency details. In practice, weighted versions of these patterns are used for texton clustering. The weight for the greyscale pattern is set to 1.0, and the weight for the feature pattern is
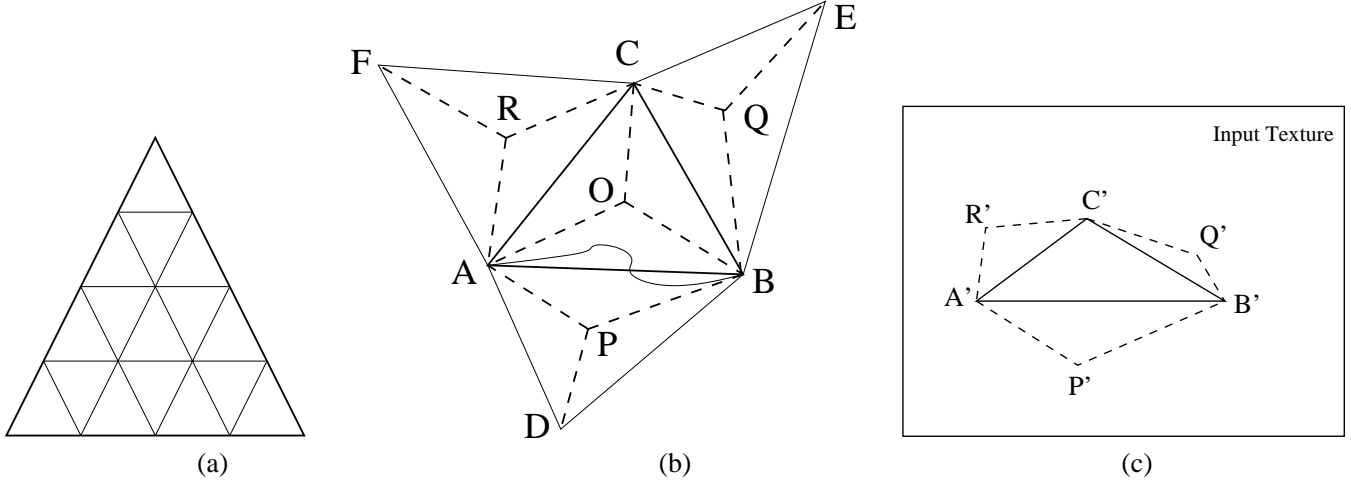
Figure 2. (a) A fine grid defined within a triangle. (b) A graphcut is performed inside the quadrilateral region $OAPB$ to obtain a refined boundary between the two adjacent texture patches. (c) An extended hexagonal texture patch corresponding to $\triangle ABC$ in (b).

set to 0.3. These weighted patterns are treated as different channels of the same texture neighborhood during texton clustering. Once we have the collection of textons, the rest of the synthesis steps follow [15].

When there are multiple input textures, every time we need to search for a texture patch for a triangle, we find the best candidate from each input texture and then choose among them the one with the highest matching score. Usually, we would like to set up for each input texture a target percentage in the output texture. To approximately control the synthesis process using these target percentages, we define a Gaussian function for each input. The standard deviation of the Gaussian is set to be the target percentage of the input texture. The function value of the Gaussian is used to modulate the matching score. When the actual percentage is lower than the target percentage, the Gaussian returns a large value which does not obviously affect the matching score; when the actual percentage is higher than the target percentage, the Gaussian returns a small value which significantly decreases the matching score. Thus, these Gaussian functions implicitly control the likelihood of sampling a specific input texture.

4 Texture Resampling and Boundary Refinement

Since we would like to perform boundary refinement on the triangular texture patches and Laplacian reconstruction over the entire synthesized texture, indexing the texture patch for each triangle as three pairs of texture coordinates in the input texture space becomes insufficient. To facilitate these later steps, we resample the texture patches
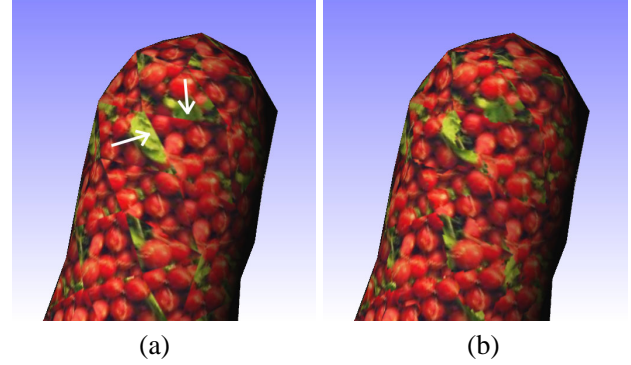

Figure 3. (a) Initial texture patch assignment result with two of the seams indicated by arrows. (b) Result obtained from boundary refinement with Graphcut.

onto a high-resolution grid over the original mesh surface. The high-resolution grid within each triangle is set up using barycentric coordinates as shown in Fig. 2(a). That is, every edge of the triangle has the same number of sample points. We actually further enforce that all edges in the triangle mesh have the same number of sample points. Thus, the subgrids within two adjacent triangles coincide on their shared edge to avoid T-junctions. If the original triangle mesh has some overly large or elongated triangles, we split those triangles in a preprocessing step while avoiding T-junctions.

We resample the texture patches previously assigned to the triangles onto this high-resolution grid. As shown in Fig. 2(b)-(c), suppose a triangle $\triangle ABC$ in the mesh has

a corresponding triangular texture patch $\Delta A'B'C'$ in one of the input texture examples. To facilitate boundary refinement at a later step, we actually resample an area larger than $\Delta A'B'C'$. Suppose ΔABD, ΔBCE and ΔCAF are the three triangles adjacent to ΔABC. Their centers are P, Q and R, respectively. We first flatten these three triangles onto the same plane where ΔABC resides and obtain the new locations of their centers. From these new locations, we can further obtain their corresponding locations P', Q' and R' in the 2D texture space. We resample the entire hexagonal area $A'P'B'Q'C'R'$ in the texture space onto the corresponding region, $APBQCR$, of the high-resolution grid. Thus, each resampled texture patch of a triangle extends into its three adjacent triangles. Suppose the center of ΔABC is O. During this resampling, ΔOAB, ΔOBC and ΔOCA not only obtain color values from the texture patch originally assigned to ΔABC, but also obtain a second color value from the extended hexagonal patches corresponding to the three adjacent triangles of ΔABC.

During initial texture patch assignment discussed in the previous section, each triangle is assigned a triangular texture patch. The boundary between two adjacent texture patches coincide with the shared edge of their corresponding triangles. In a subsequent boundary refinement procedure, we apply the graphcut algorithm in [12] to refine the boundaries between resampled texture patches on the high-resolution grid. We need to take into account the extended hexagonal texture patches to perform this procedure. Consider triangles ΔABC and ΔABD in Fig. 2(b). Suppose their hexagonal texture patches are HTP_O and HTP_P, respectively. These two texture patches have an overlapping quadrilateral region, $OAPB$. We set up a minimum graph cut problem as follows. The grid points closest to OA and OB are constrained to have colors from the texture patch HTP_O while the grid points closest to PA and PB are constrained to have colors from the patch HTP_P. The vertices A and B also have fixed colors. We then seek a minimum graph cut between A and B and within the region $OAPB$. The algorithm in [12] is applied to find this cut which can provide a better transition between the high frequency details of the two texture patches than the original boundary. The grid points falling on the same side of the cut as O obtain colors from patch HTP_O while the grid points on the other side of the cut obtain colors from patch HTP_P. Note that we still use the rectified textures during boundary refinement because the original texture colors may have large discontinuities along the triangle edges, which prevent the graphcut algorithm to find a different cut that provides better transition for high frequency details. Fig. 3(a)-(b) demonstrate the effectiveness of this boundary refinement procedure.

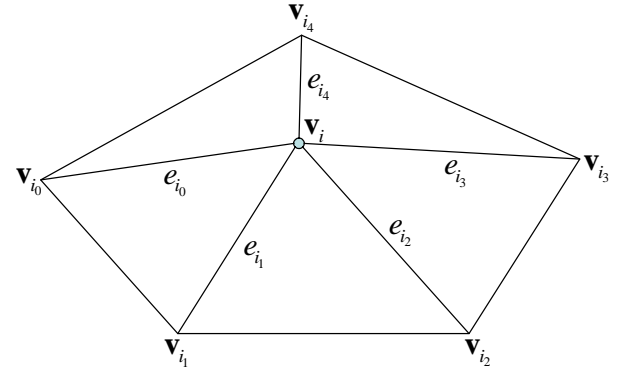Since the colors of the mesh vertices are not refined at all during the aforementioned boundary refinement, we

Figure 4. The 1-ring structure of a vertex, $\mathbf{v}_i$.

also designed another graphcut procedure specifically tailored for them. We first define an umbrella region centered at each vertex, and then flatten that region onto a parameterization plane. A subsequent graph cut is performed in this flattened region to refine the boundaries of the texture patches there. However, in our experiments, we have not observed any obvious improvements in visual quality due to this vertex-centric refinement. Therefore, we leave it as an optional step.

5 Laplacian Texture Reconstruction

The synthesis process in the previous sections focuses on high frequency details. We call the surface texture synthesized by the previous steps the *intermediate* texture. There are obvious seams inbetween adjacent texture patches in the intermediate texture because of discontinuities in the low frequency components. To remove these large discontinuities while still preserving high frequency texture details, we present a texture reconstruction technique based on the Laplacian operator which encodes high frequency features. Given the estimated Laplacians of the intermediate texture, the reconstruction process tries to obtain a new continuous surface texture which can reproduce the Laplacians. The reconstruction process uses the high-resolution grid previously generated for texture resampling.

5.1 A Weighted Laplacian Operator

The Laplacian of a vertex $\mathbf{v}_i$ in the high-resolution grid is computed by collecting the colors of its 1-ring neighbors as shown in Fig. 4. To compensate the non-uniform shape of the triangles, Fujiwara weights [8] are used:

$$L(\mathbf{v}_i) = - \sum_{0 \le j < N(i)} \frac{1}{e_{i_j}} (c_i - c_{i_j}), \qquad (1)$$

where $\mathbf{v}_{i_j}$ is a vertex directly connected to $\mathbf{v}_i$, e_{i_j} represents the edge length between $\mathbf{v}_i$ and $\mathbf{v}_{i_j}$, c_i and c_{i_j} represent

the colors at the vertices, and $N(i)$ represents the number of neighboring vertices of $\mathbf{v}_i$. This Laplacian operator can also be rewritten as:

$$L(\mathbf{v}_i) = - \sum_{0 \le j < N(i)} L^f(\mathbf{v}_i, \mathbf{v}_{i_j}, \mathbf{v}_{i_{j+1}}), \quad (2)$$

where $L^f(\mathbf{v}_i, \mathbf{v}_{i_j}, \mathbf{v}_{i_{j+1}}) = \frac{1}{2e_{i_j}}(c_i - c_{i_j}) + \frac{1}{2e_{i_{j+1}}}(c_i - c_{i_{j+1}})$. $L^f(\mathbf{v}_i, \mathbf{v}_{i_j}, \mathbf{v}_{i_{j+1}})$ only involves the three vertices of a triangular face in the 1-ring structure of $\mathbf{v}_i$. This formulation allows us to consider the Laplacian operator as a summation of these facewise terms.

Since we would like to remove discontinuities along patch boundaries but still preserve original high frequency features within the texture patches, it is desirable to have spatially adaptive smoothing. To achieve this goal, we designed a weighted Laplacian operator which imposes potentially different weights on the edges. Eqs. (1) and (2) thus become

$$L_w(\mathbf{v}_i) = - \sum_{0 \le j < N(i)} \frac{w_{i_j}}{e_{i_j}}(c_i - c_{i_j}) \quad (3)$$

$$= - \sum_{0 \le j < N(i)} L^f_w(\mathbf{v}_i, \mathbf{v}_{i_j}, \mathbf{v}_{i_{j+1}}), \quad (4)$$

where w_{i_j} is a positive weight for the edge between $\mathbf{v}_i$ and $\mathbf{v}_{i_j}$, and $L^f_w(\mathbf{v}_i, \mathbf{v}_{i_j}, \mathbf{v}_{i_{j+1}})$ is also a facewise term similar to $L^f(\mathbf{v}_i, \mathbf{v}_{i_j}, \mathbf{v}_{i_{j+1}})$. If both $\mathbf{v}_i$ and $\mathbf{v}_{i_j}$ are from the same texture patch, we simply set $w_{i_j} = 1$; otherwise, w_{i_j} can be either smaller or larger than 1. If the weight of an edge is less than 1, the bonding between the two vertices of the edge is weakened, and there is less smoothing across the edge. Too small a weight may increase the stiffness of the resulting linear system discussed in the next section.

5.2   Laplacian Reconstruction

Given the Laplacians of the intermediate texture, we would like to reconstruct a new texture with the same Laplacians. Therefore, we set up a linear system with one equation per vertex. The equation for vertex $\mathbf{v}_i$ is expressed as

$$- \sum_{0 \le j < N(i)} \left(\frac{w_{i_j}}{2e_{i_j}}(c_i - c_{i_j}) + \frac{w_{i_{j+1}}}{2e_{i_{j+1}}}(c_i - c_{i_{j+1}})\right) = L_i, \quad (5)$$

where c_i and c_{i_j} represent unknown vertex colors in the new texture we would like to solve and L_i represents the estimated Laplacian of the intermediate texture at $\mathbf{v}_i$ using Eq. (3). The left hand side of this equation is actually the weighted Laplacian of the unknown new texture at $\mathbf{v}_i$. Since the weighted Laplacian is a linear operator, this equation is a linear equation of the unknown vertex colors. Note that if the textures have three color channels, there are three equations for each vertex. The collection of equations

for all the vertices form a sparse linear system which has a symmetric coefficient matrix. Since the Laplacian operator is translation invariant, we need to fix the color of at least one vertex in order to obtain a unique solution of the linear system. Such fixed colors essentially form a boundary condition of the equations. Efficient iterative solvers [17] are a good choice for such a sparse linear system. In practice, we use a preconditioned (Incomplete Choleskey Factorization) conjugate gradient method [11].

5.2.1   Laplacian Estimation at Patch Boundaries

There are additional details concerning the estimation of the right hand side of Eq. (5) since the intermediate texture consists of patches with discontinuities on their boundaries. According to Eq. (3), the weighted Laplacian of a vertex can be estimated by accumulating a simpler term, $L^f_w(\mathbf{v}_i, \mathbf{v}_{i_j}, \mathbf{v}_{i_{j+1}})$, over all the triangular faces surrounding the vertex. However, a triangle may stride two or more texture patches. We summarize the estimation of this term as follows.

- If $\mathbf{v}_i$, $\mathbf{v}_{i_j}$ and $\mathbf{v}_{i_{j+1}}$ belong to the same patch, we directly use their colors to estimate $L^f_w(\mathbf{v}_i, \mathbf{v}_{i_j}, \mathbf{v}_{i_{j+1}})$.

- If the three vertices belong to two different patches, we should not directly use their existing colors because there may be a large gap among them. Since there must be a dominant patch having two of the three vertices, during the estimation of $L^f_w(\mathbf{v}_i, \mathbf{v}_{i_j}, \mathbf{v}_{i_{j+1}})$, the color of the third vertex should be taken from an extended version of the dominant patch to avoid large gaps.

- If the three vertices belong to three different patches, we simply randomly choose a dominant patch from the three. The colors of the other two vertices are taken from an extended version of that dominant patch.

5.2.2   Global Reconstruction

When the average brightness and color of the patches in the intermediate texture differ significantly (e.g. when they are from different complicated textures), we set up a sparse boundary condition and simultaneously solve the system of equations in (5) to remove the differences. As mentioned earlier, the boundary condition should consist of at least one constraint on the variables. A simple equality constraint is declared by setting the color of a vertex to be a fixed value. Such constraints reduce the number of variables in the linear system. The reduced linear system has a unique solution. The user can choose to interactively specify such constraints. In the absence of user-defined constraints, our program chooses to fix the colors at the centers of a random subset of the patches in the mesh. A more sophisticated constraint is defined by setting a linear combination of the

vertex colors to be a fixed value. For example, we experimented with setting the average of all vertex colors to be a fixed value. Such linear constraints can be integrated into the linear system by considering them as additional equations. When there is exactly one linear constraint, the resulting enhanced linear system has a unique solution which defines a globally continuous new surface texture. When there is more than one linear constraints, the system becomes overdetermined, and a least-squares solution should be obtained.

5.2.3 Local Reconstruction

When the average brightness and color of the patches in the intermediate texture are similar (e.g. when they are all from the same sample texture), locally reconstructing the texture can produce good results with much less computational cost than the global reconstruction. This is done by imposing color constraints on all the boundary vertices of the texture patches. The constrained color for every boundary vertex $\mathbf{v}_i$ is computed by simply blending the existing colors in its 1-ring structure,

$$c_{fixed}(\mathbf{v}_i) = \frac{1}{N(i)} \sum_{0 \le j < N(i)} c_{i_j}, \qquad (6)$$

where the neighboring vertices of $\mathbf{v}_i$, $\{\mathbf{v}_{i_j}\}$, may belong to different patches. These dense color constraints effectively disconnect the texture patches from each other. The resulting linear system can be solved patch by patch. In practice, this scheme is a few times faster than the global reconstruction. Nevertheless, it only propagates information within each texture patch, which makes it better than local feathering along the patch boundaries but prevents it from resolving color differences on patches that are remote to each other. Therefore, this local scheme provides a trade-off between quality and efficiency.

Fig. 5 demonstrates the visual quality of both local and global texture reconstruction. In the intermediate textures, there are obvious seams among the patches due to differences in low frequency components. Local Laplacian reconstruction can certainly remove these seams and create smooth transitions among the patches. However, it fails to produce large-scale changes that would make the base colors of the patches more consistent. On the other hand, global Laplacian reconstruction can perform such large-scale changes and produce more desirable results. To fully test the capability of global reconstruction, in the last example shown in Fig. 5, we artificially add a large random color shift to every texture patch in the intermediate texture. Global reconstruction can successfully remove these large color shifts and recover a consistent base color for all the patches. The reconstructed surface texture appears similar to the original input texture.

	# mesh vertices/faces	# grid vertices/faces
Bunny	2503 / 5002	640258 / 1280512
Camel	2444 / 4884	625154 / 1250304
Pawn	510 / 1016	130050 / 260096
V-shape	170 / 336	43010 / 86016

Table 1. The number of vertices and faces of the original meshes and their corresponding fine grids used in this paper.

	Initial	Refinement	Local / Global Lapl
Bunny	8	5	13 / 25
Camel	3	4	13 / 28
Pawn	< 1	1	2 / 4.5

Table 2. The average running times (in seconds) of different stages of our algorithm on three meshes. These times were measured on a 3.2GHz AMD processor. "Initial" refers to the initial texture patch assignment stage; "Refinement" refers to the texture resampling and boundary refinement stage; "Local/Global Lapl" refers to local and global Laplacian texture reconstruction.

6 Additional Experimental Results

We have conducted a large number of experiments on surface texture synthesis and mixing using the algorithm developed in this paper. Besides the examples shown in Fig. 5, we show a few additional results in Fig. 6. The first example in Fig. 6 gives a good demonstration on the fact that our algorithm can significantly improve the variability of the synthesized textures. The original FLOWERS texture has too few color variations. Extending such a texture over a large surface area would not make the result very appealing. By mixing it with the two leaf textures, the synthesized results become more interesting. In the first row of Fig. 6, the left one is the local reconstruction result while the right one is the global reconstruction result. In this particular case, both of them look interesting. The local result retains the rich colors of the three input textures while the global result has a smooth and subtle color change over the entire mesh.

The statistics of the meshes used in this paper are summarized in Table 1. The running times of various stages of our algorithm are also summarized in Table 2.

7 Conclusions

In this paper, we proposed to decompose texture synthesis into two relatively disjoint stages. In the first stage, an in-

termediate synthesized texture is generated by only considering the high frequency details during neighborhood search and matching. In the second stage, we perform Laplacian texture reconstruction which retains the high frequency details but computes consistent low frequency components. It does not only affect texels close to discontinuities, but also modifies the rest of the texels. Therefore, it can be viewed as a global feature-preserving smoothing step, and is more effective than local feathering. Experiments indicate that our two-stage synthesis can produce desirable results for regular texture synthesis as well as texture mixing from multiple sources. In future, we would like to implement Laplacian reconstruction and other time-consuming steps on GPUs to achieve interactive performance.

References

[1] M. Ashikhmin. Synthesizing natural textures. In *ACM Symposium on Interactive 3D Graphics*, pages 217–226, 2001.

[2] Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):120–135, 2001.

[3] P. Bhat, S. Ingram, and G. Turk. Geometric texture synthesis by examples. In *Eurographics Symposium on Geometry Processing*, 2004.

[4] J. D. Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proc. of Siggraph*, pages 361–368, 1997.

[5] J. Dischler, K. Maritaud, B. Levy, and D. Ghazanfarpour. Texture particles. *Computer Graphics Forum*, 21(3):401–410, 2002.

[6] A. Efros and W. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH'01*, pages 341–346, 2001.

[7] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *Intl. Conf. Computer Vision*, pages 1033–1038, 1999.

[8] K. Fujiwara. Eigenvalues of laplacians on a closed riemannian manifold and its nets. In *Proceedings of the American Mathematical Society*, pages 123:2585–2594, 1995.

[9] D. Heeger and J. Bergen. Pyramid-based texture analysis/synthesis. In *Proc. of SIGGRAPH*, pages 229–238, 1995.

[10] A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, and D. Salesin. Image analogies. In *SIGGRAPH'01*, pages 327–340, 2001.

[11] D. Kershaw. The incomplete cholesky–conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26(1):43–65, 1978.

[12] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003.

[13] L. Liang, C. Liu, Y. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis using patch-based sampling. *ACM Trans. Graphics*, 20(3):127–150, 2001.

[14] Y. Liu, W.-C. Lin, and J. Hays. Near-regular texture analysis and manipulation. *ACM Transactions on Graphics*, 23(3):366–374, 2004.

[15] S. Magda and D. Kriegman. Fast texture synthesis on arbitrary meshes. In *Eurographics Symposium on Rendering*, pages 82–89, 2003.

[16] P. Perez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Trans. on Graphics*, 22:313–318, 2003.

[17] Y. Saad. *Iteragive Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.

[18] E. Simoncelli and J. Portilla. Texture characterization via joint statistics of wavelet coefficient magnitudes. In *Fifth Intl. Conf. on Image Processing, Vol.1*, pages 62–66, 1998.

[19] C. Soler, M.-P. Cani, and A. Angelidis. Hierarchical pattern mapping. In *SIGGRAPH'02*, pages 673–680, 2002.

[20] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Symposium of Geometry Processing*, 2004.

[21] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proc. Intl. Conf. on Computer Vision*, pages 836–846, 1998.

[22] G. Turk. Texture synthesis on surfaces. In *SIGGRAPH'01*, pages 347–354, 2001.

[23] L.-Y. Wei. *Texture Synthesis by Fixed Neighborhood Searching*. PhD thesis, Stanford University, 2001.

[24] L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of Siggraph*, pages 479–488, 2000.

[25] L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH'01*, pages 355–360, 2001.

[26] Q. Wu and Y. Yu. Feature matching and deformation for texture synthesis. *ACM Transactions on Graphics (special issue for SIGGRAPH 2004)*, 23(3):362–365, 2004.

[27] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics (special issue for SIGGRAPH 2004)*, 23(3):641–648, 2004.

[28] S. Zelinka and M. Garland. Jump map-based interactive texture synthesis. *ACM Transactions on Graphics*, 23(4):929–1073, 2004.

[29] J. Zhang, K. Zhou, L. Velho, B. Guo, and H.-Y. Shum. Synthesis of progressively-variant textures on arbitrary surfaces. In *SIGGRAPH'03*, pages 295–302, 2003.

[30] S. Zhu, Y. Wu, and D. Mumford. Filters, random fields and maximum entropy (frame)-towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.
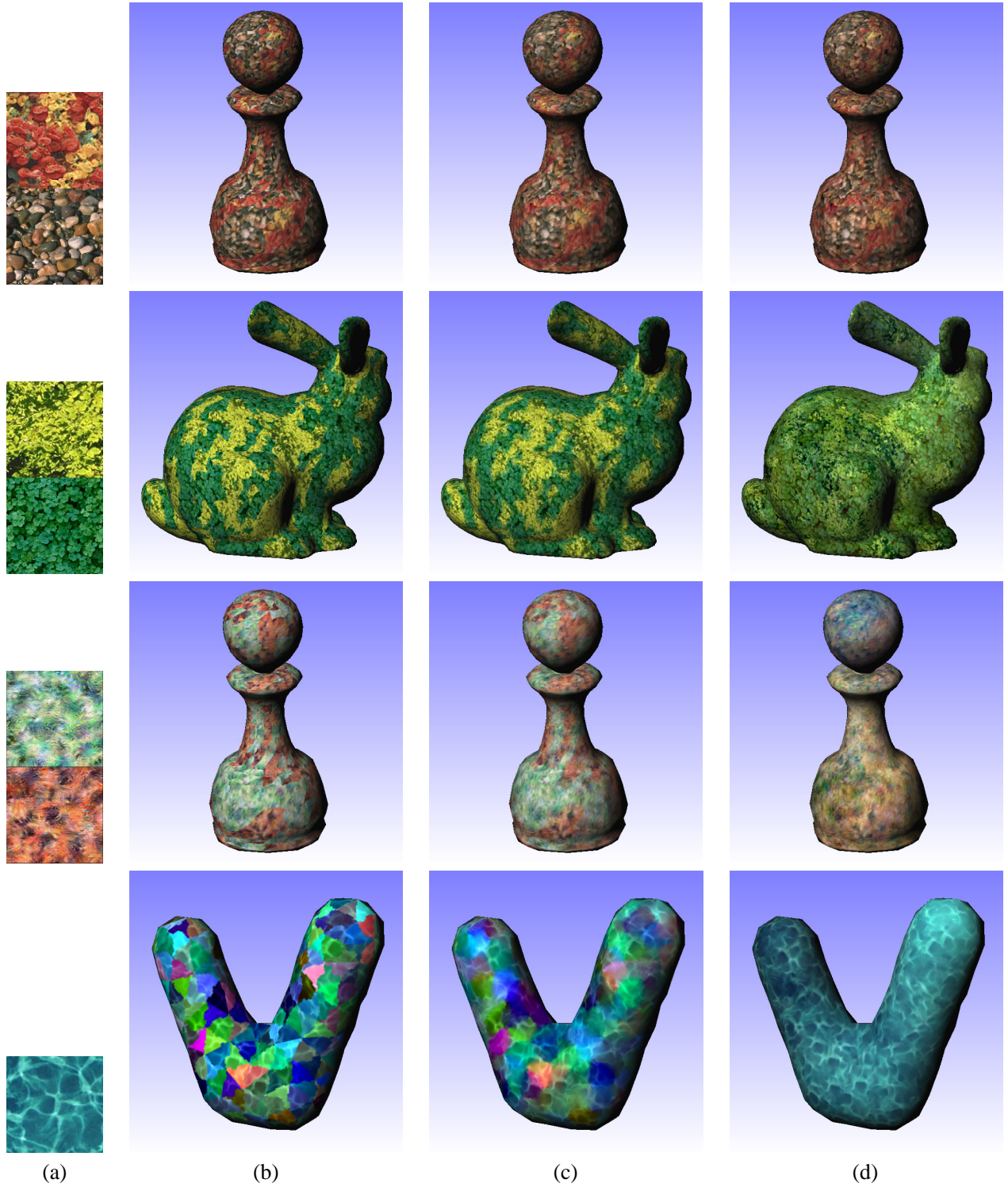
Figure 5. (a) Input texture examples. **(b)** Synthesized intermediate textures with color discontinuities among patches. **(c)** Textures computed from local Laplacian reconstruction. **(d)** Textures computed from global Laplacian reconstruction. Note that local reconstruction works reasonably well for the texture mixture in the first row because the colors of the mixed texture patches are not too different. However, for the remaining three mixture examples, global reconstruction produces more natural and consistent low frequency components. The intermediate texture in the last row is artificially modified by adding a random color shift to each texture patch. Global texture reconstruction can successfully remove such color shifts.
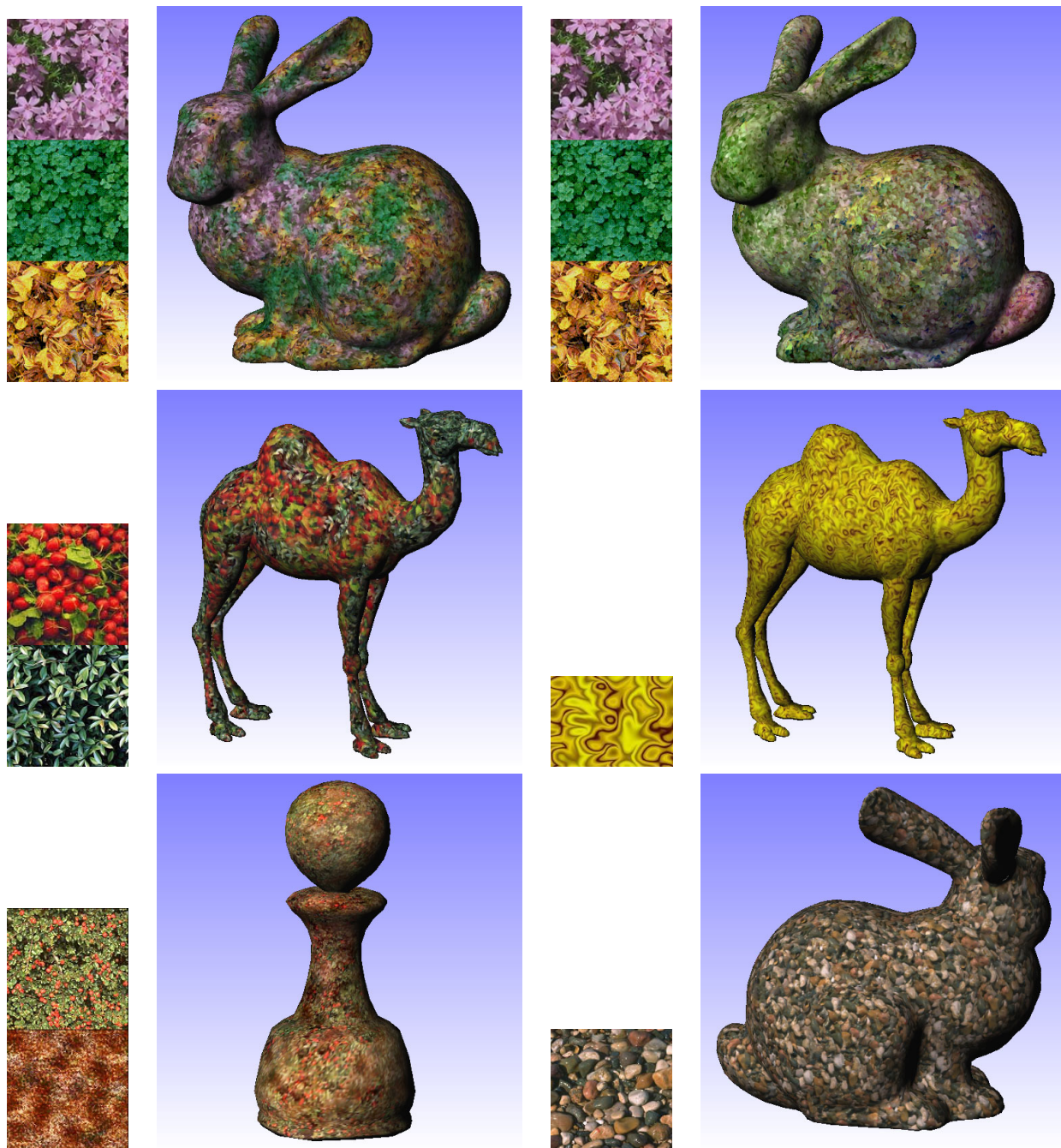
Figure 6. Additional surface texture synthesis and mixing results.